動的再構成可能プロセッサ向け仕様記述言語の開発と実問題の適用 (アルゴリズムと計算機科学の数理的基盤とその応用)

AUTHOR(S):

中居, 佑輝; 山根, 智

数理解析研究所講究録
第1691巻 2010年 8-14

動的再構成可能プロセッサ向け仕様記述言語の開発と実問題の適用

金沢大学自然科学研究科電子情報工学専攻　中居　佑輝 (Yuki Nakai)　山根　智 (Satoshi Yamane)
Division of Electrical and Computer Engineering,
Kanazawa University Graduate School of Natural Science & Technology

1　はじめに

1.1　研究の背景

本研究は動的再構成可能プロセッサ [1] を搭載した組込みシステムの仕様記述言語を提案する．動的再構成可能プロセッサとは，FPGAのように論理構成を再構成できるプロセッサである．FPGAとの違いは，瞬時に再構成できるため処理の途中に動的に再構成していくことが可能な点である．動的再構成可能プロセッサでは，以下のものが実現されている．

- ある機能を実現するためのタスクの内部処理を，複数のコンテキスト（論理回路構成）に分解して，同一基盤上でコンテキストを切り替えながら少ない面積で処理を行う [2]
- コンテキスト毎にクロック周波数を変更する [3]
- チップ上で同時に複数のタスクを実行する [4]

上記のことより，動的再構成可能プロセッサを搭載したシステムでは，複数のタスクを同時に実行させ，かつその時々の回路構成に最適な周波数へと変化させていくような用い方が予想される．この方式では，動的再構成可能プロセッサ上で動作するタスクは，同時に実行する他のタスクの周波数や，待ち行列の中身により，毎回違う応答時間を持つこととなる．これでは，設計するシステムがリアルタイム性を満たすかどうかを判断することが難しい．本研究は，このようなシステムの設計言語（仕様記述言語）を提案する．提案する言語は操作的な意味を持っており，状態列を網羅的に探索する検証器や，ある状態列を視覚的にみるシミュレータの開発が行える．これにより，リアルタイム性を満たすかどうかの判定が可能となる．また，動的再構成可能プロセッサに特化した仕様記述言語は本研究が初めてである．

1.2　仕様記述言語の関連研究

動的再構成可能システムの既存の仕様記述言語としては，リアクティブモデル，リアルタイムモデル，ハイブリッドモデルのいずれかのモデルがあり，仕様記述のスタイルとして，プロセス代数，オートマトン，ペトリネットがある．代表的なものとしては以下がある．カリフォルニア大学バークレー校 A. Deshpande らの SHIFT[5] 及びオランダのアイントホーヘン大学 F. Kratz らの，Charon[6] を拡張した R-Charon[7] は，ハイブリッドネットワークの構成要素間の通信リンクを動的に生成・消滅させて，ネットワークの構造が動的に変化するシステムを仕様記述できる手法であり，バイオケミカルプロセス，輸送システムやモジュラ再構成可能ロボットなどがターゲットである．また，Charon のシミュレータ [6] が開発されている．しかし，これらはイベント駆動の処理が記述できないために，動的再構成可能プロセッサを仕様記述できない．また，MIT の N. Lynch らは，リアクティブオートマトンである Input/Output Automata を動的なオートマトンの生成消滅で拡張した仕様記述言語を開発している [8]．DRP のモデル化に関する関連研究としては，J. Teich らや潮らがリアクティブオートマトンを用いて LSI を離散モデル記述した例がある [9]．

本研究は，R-Charon をベースに，イベント同期によるバスの同期通信の記述手法を導入し，そのセマンティクスをシミュレータや形式的検証を可能にする操作的意味に変更した言語を開発するものである．

2　仕様記述言語の提案

提案する言語の概念は以下である (図 1)．

System
　システム全体を表す．

Class
　システムを構成するオブジェクトの構造を表す．

Object
　システムを構成するオブジェクトを表す．

Mode
　オブジェクトのモード (動作) を表す．

2.1　構文

定義 1 ($System$ の定義)
　システム全体を $System =< Class, Object, Obj_0 >$ と定義する．図 1 のように，クラスと，そのクラスから生成されたある時点のオブジェクトから

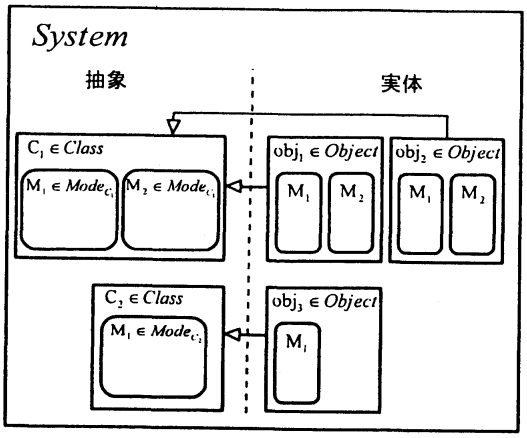

図 1: 概念図

成る.

- *Class* は構造の有限集合
- *Object* はオブジェクトの有限集合
- Obj_0 は初期オブジェクトの有限集合 □

定義 2 (*Class* の定義)
C∈ *Class* のとき,
クラス C を C=< $Mode, V_c, V_d, V_r, Act, I$ > と定義する. 図 2 のように, クラス内で使用する変数とアクション, 動作で構成される. また, クラスは角の尖った枠の中に記述する.

- *Mode* は動作を表すオートマトンの有限集合
- V_c はクロック変数の有限集合
- V_d は離散変数の有限集合
  グローバル変数 V_{d_g} とローカル変数 V_{d_l} の和集合である.
- V_r はオブジェクトを指す参照変数の有限集合
- *Act* はアクションの有限集合
  入力アクションの有限集合 Act_{in} と出力アクションの有限集合 Act_{out} の和集合である.
- I は各変数の初期値の有限集合
  オブジェクトを生成する際に与えられる変数の初期値の集合である. オブジェクト生成時に明示的に初期値が与えられるが, それ以外の変数は I の値が初期値として代入される. V_c = $\{v_{c_1}, v_{c_2}, ..., v_{c_n}\}$, V_d = $\{v_{d_1}, v_{d_2}, ..., v_{d_n}\}$, V_r = $\{v_{r_1}, v_{r_2}, ..., v_{r_n}\}$ のとき I = $\{v_{c_1} := iv_{c_1}, ..., v_{c_n} := iv_{c_n}, v_{d_1} := iv_{d_1}, ..., v_{d_n} := iv_{d_n}, v_{r_1} := \epsilon, ..., v_{r_n} := \epsilon\}$ のように書く. なお, 参照変数の ϵ は参照先がないことを表す. □

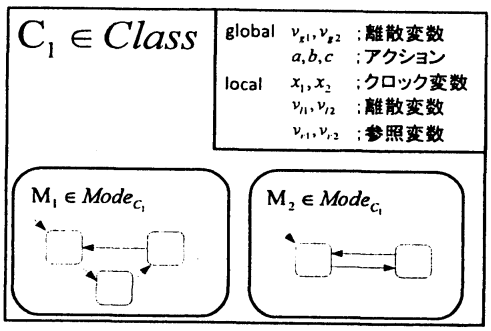

図 2: Class

定義 3 (*Object* の定義)
図 3 のように, C∈ *Class* の構造を持つオブジェクトが obj∈ *Object* であるとき, オブジェクト obj を obj=< $Mode_{\text{obj}}, V_{c_{\text{obj}}}, V_{d_{\text{obj}}}, V_{r_{\text{obj}}}, Act_{\text{obj}}$ > と定義する. オブジェクトは角の尖った枠の中に記述する.

- $Mode_{\text{obj}}$ は動作を表すオートマトンの有限集合
  クラス C の *Mode* のインスタンスである. $Mode_{\text{obj}}$ の要素が複数ある場合はそれらのオートマトンは並列に動作する. 図 3 の例では, M_1,$M_2 \in Mode_{\text{obj}}$ である.
- $V_{c_{\text{obj}}}$ はクロック変数の有限集合
  クラス C の V_c のインスタンスである.
- $V_{d_{\text{obj}}}$ は離散変数の有限集合
  クラス C の V_d のインスタンスである. グローバル変数 $V_{d_{obj_g}}$ とローカル変数 $V_{d_{obj_l}}$ がある.
- $V_{r_{\text{obj}}}$ は参照変数の有限集合
  クラス C の V_r のインスタンスである. obj,obj_1 ∈ *Object*, v_{r_0} ∈ $V_{r_{\text{obj}}}$, v_{r_0} が obj_1 を指し, obj_1 のグローバル変数が v_{g_0} のとき, obj から obj_1 のグローバル変数を $v_{r_0}.v_{g_0}$ として読み書きできる. また, V_r = $\{v_{r_0}, v_{r_1}, ..., v_{r_n}\}$, V_d = $\{v_{d_0}, v_{d_1}, ..., v_{d_n}\}$ であるとき, $V_r.V_d$ = $\{v_{r_0}.v_{d_0}, v_{r_0}.v_{d_1}, ..., v_{r_0}.v_{d_n}, v_{r_1}.v_{d_0}, v_{r_1}.v_{d_1}, ..., v_{r_n}.v_{d_n}\}$ と記述する.
- Act_{obj} はアクションの有限集合

クラス C の Act のインスタンスである．入力アクション $Act_{\mathrm{obj}_{in}}$ と出力アクション $Act_{\mathrm{obj}_{out}}$ がある．$a? \in Act_{\mathrm{obj}_{in}}$，$a! \in Act_{\mathrm{obj}_{out}}$ のとき，$a!$ が出力されると対応する $a?$ を持つエッジの遷移が引き起こされる．また，参照変数を用いて $v_{r_0}.a!$ のように外部のオブジェクトからアクションを出力することができる． □

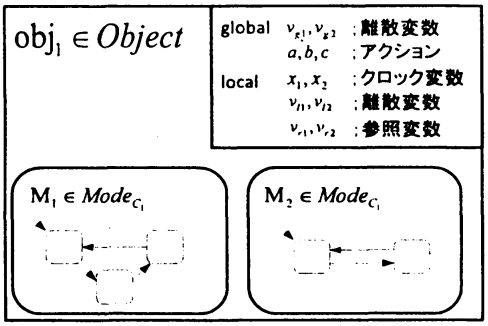

図 3: Object

定義 4 (Mode の定義)

obj∈ $Object$ の動作を表すオートマトンが M∈ $Mode_{\mathrm{obj}}$ であるとき，図 4 のようにオートマトン M を M=< $L, h, type, l_0, V_{c_M}, V_{d_M}, V_{r_M}, D, Inv, Act_M, T$ > と定義する．Mode は角の丸い枠の中に記述する．

- L はロケーションの有限集合
- h はサブロケーションを返す関数 $h : L \rightarrow 2^L$
- $type$ は型関数 $type : L \rightarrow \{BASIC, OR\}$
  $l \in L$ の型を返す関数．型 $BASIC$ を持つロケーションは基底ロケーションであり，$h(l) = \emptyset$ となるロケーションである．型 OR を持つロケーションは OR ロケーションであり，$h(l) \neq \emptyset$ となる，つまりサブロケーションを持つロケーションである．
- l_0 は初期ロケーション
- V_{c_M} はクロック変数の有限集合
  $V_{c_M} \subseteq V_{c_{obj}}$
- V_{d_M} は離散変数の有限集合
  $V_{d_M} \subseteq V_{d_{obj}}$
- V_{r_M} は参照変数の有限集合
  $V_{r_M} \subseteq V_{r_{obj}}$

  Mode で用いるタイミング制約などの条件式を以下で記述する．

$\phi ::= \psi \sim \gamma | \phi_1 \wedge \phi_2 | true$
ただし，
$\psi ::= x | d$
$\gamma ::= d | a | \gamma_1 + \gamma_2 | \gamma_1 - \gamma_2$
$x \in V_{c_M}$，$d \in V_{d\star}$，$V_{d\star}$ は読み書きできる離散変数の集合 $V_{d\star} = V_{d_M} \cup V_{r_M}.V_{d_{obj'_g}}$，$a \in \mathbb{R}$ は実数であり，$\sim \in \{\leq, <, >, \geq, ==\}$ である．また，全てのタイミング制約 ϕ の集合を Φ とする．

各ロケーションへ割り付ける $flow$ 条件は以下で定義する式である．

$inc ::= \dot{x} = a$
ただし，$\dot{x}$ はクロック変数 x の傾きである．また，全ての勾配 inc の集合を Inc とする．

- D はロケーションにクロック変数の傾きを割りつける関数 $D : L \rightarrow 2^{Inc}$
- Inv はロケーションに不変条件 ϕ を割り当てる関数 $Inv : L \rightarrow \Phi$
- Act_M はアクションの有限集合
  入力アクション $Act_{M_{in}} = Act_{\mathrm{obj}_{in}}$ と出力アクション $Act_{M_{out}} = Act_{\mathrm{obj}_{out}} \cup V_{r_M}.Act_{\mathrm{obj'}_{out}}$ の和集合である．

変数の算術式を以下で記述する．

$reset ::= x := 0 | i := \gamma$
ただし，$x \in V_{c_M}$，$i \in V_{d\star}$ である．また，全ての算術式 $reset$ の集合を $RESET$ とする．
$reset_1, reset_2, ..., reset_n \in RESET$ であるとき，$list_{reset}$=$[reset_1, reset_2, ..., reset_n]$ のように順序付きで並べたリストの全ての集合を $LIST_{RESET}$ とする．

オブジェクトの生成関数と破棄関数を定義する．
オブジェクトの生成関数は $Create(\mathrm{C}, Init, v_{r'})$ である．ただし，C∈ $Class$，$v_{r'} \in V_{r_{M'}}$ であり，この関数により C の構造を持つオブジェクトが，初期値が $Init$ で指定されていれば $Init$ で，指定されていない初期値があればクラスで定義されている I を初期値として生成される．また，生成したオブジェクトのポインタが参照変数 $v_{r'}$ に格納される．
オブジェクトの破棄関数は $Destroy(v_r)$ である．ただし，$v_r \in V_{r_M}$ であり，この関数により v_r

が指すオブジェクトを破棄する．これら関数を以下のように記述する．

$ocd ::= Create(\mathrm{C}, Init, v_{r'}) | Destroy(v_r)$
全ての ocd の集合を OCD とする．

- T はエッジの有限集合

  $T \subseteq L \times Act_{\mathrm{M}_{in}} \times \Phi \times LIST_{RESET} \times OCD \times Act_{\mathrm{M}_{out}} \times L$
  エッジには複数の算術式を割り当てることができる．それらはエッジに割り当てられている順に評価される． □

  図 4 の例では，クロック変数 x_1 の勾配は，ロケーション l_4 で 2，それ以外のロケーションで 1 である．初期ロケーションは l_0 であり，起動から x_1 が 10 となるまで留まり，l_3 の l_4 へと遷移する．l_4 から l_5，l_1 から l_0 への遷移は外部の Mode から $a!$ が出力され，かつ遷移可能条件を満たすときに遷移が起きる．

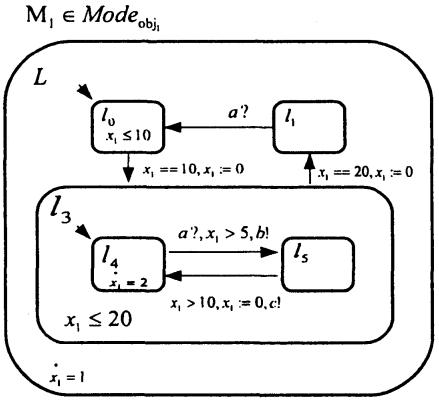

図 4: Mode

2.2 意味の定義

提案する言語の意味は，オブジェクト内の動作の意味である $Sem(\mathrm{obj})$ と，オブジェクトの生成破棄を意味する $Sem(System)$ から成る．

定義 5 ($Sem(\mathrm{obj})$ の定義)
obj$\in$ $Object$ の意味 $Sem(\mathrm{obj})$ を $Sem(\mathrm{obj})$ $=< Q_{\mathrm{obj}}, q_{\mathrm{obj}_0}, Act_\star, \rightarrow >$ と定義する．

- Q_{obj} はオブジェクトの状態の有限集合

  $q_{\mathrm{obj}} = (conf, \nu_c, \nu_d, \nu_r) \in Q_{\mathrm{obj}}$

  - $conf$ はコンフィギュレーション [10] である．
    ロケーション l のアクティブなサブロケーションを返す関数 $\rho : L \rightarrow L \bigcup \{\perp\}$ を用いて得られる，ある時点でアクティブな最大のロケーション集合 $conf = \bigcup_{k=1}^{n} \rho^*_{\mathrm{M}_k}(root)$ をコンフィギュレーションと呼ぶ．ただし，ρ は以下を満たす関数であり，$root$ はモードの最上位層のロケーション，M_1,M_2,...,$\mathrm{M}_n \in Mode_{\mathrm{obj}}$ である．

$$\rho(l) = \begin{cases} \{\perp\} \text{ (未定義)} & (type(l) == BASIC) \\ \{l'\} & (l' \in h(l),\ type(l) == OR) \end{cases}$$

$$\rho^*_{\mathrm{M}}(l) = \begin{cases} \{l\} & (type(l) == BASIC) \\ \bigcup_{l'=\rho(l)} \rho^*_{\mathrm{M}}(l') & (otherwise) \end{cases}$$

また，全てのコンフィギュレーションの集合を $CONF$ とする．図 5 はコンフィギュレーションが $\{l_1, l_2, l'_1, l'_3\}$ の例である．また，コンフィギュレーションに不変条件を割り当てる関数 $Invc$ を定義する．
$Invc : CONF \rightarrow \Phi$
$conf = \{l_0, l_1, ..., l_n\}$ であるとき，$Invc(conf) = Inv(l_0) \wedge Inv(l_1) \wedge ... \wedge Inv(l_n)$ である．

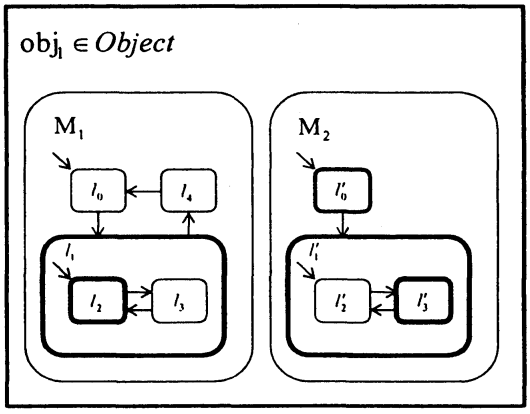

図 5: コンフィギュレーション $\{l_1, l_2, l'_1, l'_3\}$

  - $\nu_c : V_{c_{\mathrm{obj}}} \rightarrow \mathbb{R}$
    クロック変数評価．
  - $\nu_d : V_{d_{\mathrm{obj}}} \cup V_{r_{\mathrm{obj}}}.V_{d_{\mathrm{obj}_g}} \rightarrow \mathbb{R}$
    離散変数評価．
  - $\nu_r : V_{r_{\mathrm{obj}}} \rightarrow$ &obj

オブジェクトの場所(アドレス)を指す．ただし，obj$\in Object$であり，&objはobjの場所(アドレス)である．

- q_{obj_0}は初期状態
- $Act_\star$はアクションの有限集合
  $Act_\star = Act_{\mathrm{obj}} \cup V_{r_{\mathrm{obj}}}.Act_{\mathrm{obj}'_{out}}$
- $\rightarrow$は時間遷移$\rightarrow_\delta$と離散遷移$\rightarrow_d$の和集合

変数評価ν_c, ν_dに対し，$\nu_c, \nu_d \in [\![\phi]\!](\nu_c, \nu_d$が$\phi$を満たす)とは，$[\![\phi]\!]$の実数領域に$\nu_c, \nu_d$の変数評価が全て含まれていることを表す．なお，$[\![\phi]\!] : \Phi \rightarrow 2^{\mathbf{R}}$とする．

1. 時間遷移$\rightarrow_\delta$
   任意の状態$(conf, \nu_c, \nu_d, \nu_r)$に対し，$conf = conf'$，$\nu_d = \nu'_d$，$\nu_r = \nu'_r$，$\nu'_c \in [\![Invc(conf)]\!]$の時に限り，$(conf, \nu_c, \nu_d, \nu_r) \rightarrow_\delta (conf', \nu'_c, \nu'_d, \nu'_r)$である．ただし，$flow$のカーブは$f : [0, \delta] \rightarrow \mathbb{R}^n$であり，$|V_{c_{obj}}| = n$，$f(0) = \nu_c$，$f(\delta) = \nu'_c$である．

   エッジに割り当てられている変数の算術式のリスト$list_{reset}$が，順にreset_1,reset_2,...,reset_nである時，変数評価ν_c, ν_dに対し，reset_1,reset_2,...,reset_nを順番に適用した後の変数評価を$\nu_c[list_{reset}]$，$\nu_d[list_{reset}]$と記述する．

2. 離散遷移$\rightarrow_d$
   離散遷移は以下の4つの種類がある．
   - 入力ベントが無い場合
     任意の状態$(conf, \nu_c, \nu_d, \nu_r)$に対し，$conf$を構成するロケーション$l$からのエッジ$(l, \epsilon, \phi, list_{reset}, ocd, act_{out}, l') \in T$が存在し，$\nu_c, \nu_d$が$\phi$を満たし，かつ$\nu'_c[list_{reset}], \nu'_d[list_{reset}] \in [\![Invc(conf')]\!]$となるとき，
     $(conf, \nu_c, \nu_d, \nu_r) \rightarrow_d$
     $(conf', \nu'_c, \nu'_d, \nu'_r)$
     となる遷移が可能である．また，遷移時に$System$の待ち行列$list$に生成関数・破棄関数ocdを$list$の後ろから追加し，イベントact_{out}を出力する．出力されたイベントを入力に持つエッジは同期を起こす．
   - 入力ベントがある場合
     任意の状態$(conf, \nu_c, \nu_d, \nu_r)$に対し，$conf$を構成するロケーション$l$からのエッジ$(l, act_{in}, \phi, list_{reset}, ocd, act_{out}, l') \in T$が存在し，$\nu_c, \nu_d$が$\phi$を満たし，かつ$\nu'_c[list_{reset}], \nu'_d[list_{reset}] \in [\![Invc(conf')]\!]$となり，かつ$a? \in act_{in}$と対なる$a! \in Act_{obj_{out}}$が出力されたときに限り，
     $(conf, \nu_c, \nu_d, \nu_r) \rightarrow_d$
     $(conf', \nu'_c, \nu'_d, \nu'_r)$
     である．また，遷移時に$System$の待ち行列$list$に生成関数・破棄関数ocdを$list$の後ろから追加し，イベントact_{out}を出力する．出力されたイベントを入力に持つエッジは同期を起こす．
   - 外部オブジェクトの遷移による環境遷移
     任意の状態$(conf, \nu_c, \nu_d, \nu_r)$に対し，参照変数$v_r \in V_{r_{obj}}$の指す外部のオブジェクトで離散遷移が起きたとき，変数評価が更新される．
     $(conf, \nu_c, \nu_d, \nu_r) \rightarrow_d$
     $(conf', \nu'_c, \nu'_d, \nu'_r)$
     ただし，$conf = conf'$かつ$\nu_r = \nu'_r$である．$\nu'_c, \nu'_d \in [\![Invc(conf')]\!]$を満たさないときは，ただちに可能なロケーションへの遷移を行う．
   - $System$の離散遷移による環境遷移
     任意の状態$(conf, \nu_c, \nu_d, \nu_r)$に対し，オブジェクトの生成・破棄により，参照変数$v_r \in V_{r_{obj}}$の値が更新される．$Create(\mathrm{C}, Init, v_r)$によるオブジェクトの生成であれば，$v_r$に生成したオブジェクトのアドレスを格納し，$Destroy(v_r)$であれば，$V_{r_{obj}}$の変数値のうち，$v_r$と同値である変数群の値を$\epsilon$にリセットする．また，それにより，読み書き可能である変数評価ν_dも更新する．また，$conf = conf'$である．
     $(conf, \nu_c, \nu_d, \nu_r) \rightarrow_d$
     $(conf', \nu'_c, \nu'_d, \nu'_r)$

最後にオブジェクトの状態列を定義する．オブジェクトの状態列は時間遷移$\rightarrow_\delta$と離散遷移$\rightarrow_d$から成る．

$q_{\mathrm{obj}_0} \rightarrow_d q_{\mathrm{obj}_1} \rightarrow_d q_{\mathrm{obj}_2} \rightarrow_\delta q_{\mathrm{obj}_3} \rightarrow_d \cdots$

□

定義 6 ($Sem(System)$ **の定義**)
$System$ の意味 $Sem(System)$ を $Sem(System) =< Q_{sys}, q_{sys_0}, \Rightarrow >$ と定義する.

- Q_{sys} はシステムの状態の有限集合
  $q_{sys} = (AOBJ, list) \in Q_{sys}$
  - $AOBJ$ は現在存在するオブジェクトの意味の有限集合
  - $list$ は生成関数と破棄関数の待ち行列
    $list$ には実行前の生成関数と破棄関数が順番に格納される. $list_1 :: list_2$ は $list_1$ へ後方から $list_2$ を追加することを表し, $Hd(list)$ は $list$ の先頭を取りだす関数である.
- q_{sys_0} は初期状態
- $\Rightarrow$ は時間遷移 $\Rightarrow_t$ と離散遷移 $\Rightarrow_d$ の和集合

1. 時間遷移 $\Rightarrow_t$
   任意の状態 $(AOBJ, list)$ と実数 $t \in \mathbb{R}_{\geq 0}$ に対し, $list = list'$, $list$ が空である時に限り
   $(AOBJ, list) \Rightarrow_t (AOBJ', list')$
   である. ただし, $AOBJ'$ は $AOBJ$ 内の全てのオブジェクトを時間遷移させたオブジェクトの意味の集合である.
2. 離散遷移 $\Rightarrow_d$
   任意の状態 $(AOBJ, list)$ に対し, $list$ が空でない時, $list$ の先頭の要素に対し以下の遷移を行う.
   - $list$ の先頭が生成関数のとき
     $list$ の先頭の要素が生成関数 $Create(\mathrm{C}, Init, v_{r'})$ である時, $\mathrm{C} \in Class$ の構造を持ち, 初期値が $Init$ であるオブジェクトを生成し, その場所 (アドレス) を $v_{r'}$ に格納する. ただし, $v_{r'}$ は $list$ に生成関数を格納したオブジェクトが持つ参照変数である. 生成後に $list$ の先頭から $Create(C, Init, v_{r'})$ を消去する.
     $(AOBJ, list) \Rightarrow_d$
     $(AOBJ \cup \{Sem(\mathrm{obj}_{crt})\}, list')$
     ただし, $list = Hd(list) :: list'$ であり, 生成するオブジェクトは obj_{crt} である.
   - $list$ の先頭が破棄関数のとき
     $list$ の先頭の要素が破棄関数 $Destroy(v_r)$ である時, v_r が指すオブジェクト obj_{dst} を破棄する. 破棄後に $list$ から $Destroy(v_r)$ を消去する.
     $(AOBJ, list) \Rightarrow_d$
     $(AOBJ \setminus \{Sem(\mathrm{obj}_{dst})\}, list')$
     ただし, $list = Hd(list) :: list'$ である.

最後にシステムの状態列を定義する. システムの状態列は時間遷移 $\Rightarrow_t$ と離散遷移 $\Rightarrow_d$ から成る.
$q_{sys_0} \Rightarrow_d q_{sys_1} \Rightarrow_d q_{sys_2} \Rightarrow_t q_{sys_3} \Rightarrow_d \ldots$

□

定義 7 (履歴)
提案言語はステートチャート [12] と同様の履歴を持つ. 階層を持つロケーションは, 上位ロケーションを介した遷移を行う時にアクティブなロケーションの履歴を保持するための履歴状態指示子を持つことができる. 履歴状態指示子は H を丸で囲んだアイコンで表示され, このアイコンがあるロケーションに遷移してきた時は前回アクティブだったロケーションに遷移を行う. 複数の階層にわたって記憶する履歴状態指示子 (ディープヒストリー) は H に*を付けた形で表示する. □

3 動的再構成可能プロセッサ搭載システムの仕様記述

提案言語を動的再構成可能プロセッサに適用する. このモデルにより, CPU での処理 (ソフトウェア) と動的再構成可能プロセッサでの処理 (ハードウェア) の協調動作を表すことができるようになり, システム開発の全体をとらえることができるようになる. また, このモデルの検証手法・シミュレーションツールを開発することにより, 設計段階でのミスを事前に防ぐことを可能とする.

CPU タスクのスケジューリングポリシィは, プリエンプションありの優先度順スケジューリングである. 動的再構成可能プロセッサのタスクは CPU のタスクに呼び出され, 必要に応じた処理を行い, その結果を呼び出した CPU のタスクに返す. その間呼び出し元の CPU のタスクは I/O 待ち状態となる. 動的再構成可能プロセッサは 1 クロックで構成を変化できるものであり, 基盤面積さえ足りていれば, 同時に複数のタスクを処理することが可能である. その際, 最大遅延に合わせたクロック周波数に適宜変更される. 仕様記述した全体構造は図 6 である. な

お，紙面の都合上詳細なモデルの仕様記述は本稿では省略する．

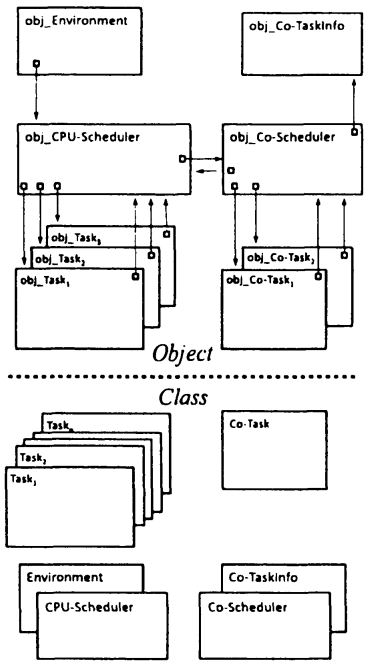

図 6: 全体構造

4 まとめと今後の課題

動的再構成可能な組込みシステムを，ソフトとハードの両面からアプローチして記述できるような言語を開発した．また，今回仕様記述したモデルを静的なハイブリッドオートマトンに変換し，HyTECH[13]を使用してタスク検証を行う実験が金沢大学の南らによってなされている [14]．その結果，静的な言語のまま検証を行うと状態爆発を起こす．今後，この言語からの動的検証理論を開発することで，記述された仕様からシームレスに検証を行うことが課題である．

参考文献

[1] 末吉，天野．リコンフィギャラブルシステム．オーム社，2005.

[2] 桂，他．リアルタイム組込みシステムの動的再構成可能プロセッサへの一実装方法の提案．IEICE technical report Vol.105, No.450, 2005, pp. 31-36.

[3] 天野，他．動的リコンフィギャラブルプロセッサにおける可変クロック機構の導入．電子情報通信学会技術研究報告 CPSY2004-65, Vol.104, No.589, 2005, pp.13-16.

[4] V. M. Tuan, et al. Performance Evaluation of Hardware Multi-process Execution on the Dynamically Reconfigurable Processor. IEICE technical report Vol.106, No.247, 2006, pp. 25-30.

[5] A. Deshpande, et al. The SHIFT programming language and run-time system for dynamic networks of hybrid systems. IEEE Transactions on Automatic Control 43(4), 1998, pp. 584-587.

[6] Y.Hur and I.Lee. Distributed Simulation of Multi-Agent Hybrid Systems. IEEE International Symposium on Object-Oriented Real-time distributed Computing (ISORC), 2002, pp.356-364.

[7] F. Kratz, et al. R-Charon, a Modeling Language for Reconfigurable Hybrid Systems. LNCS 3927, 2006, pp. 392-406.

[8] P. C. Attie and N. A. Lynch. Dynamic Input/Output Automata: A Formal Model for Dynamic Systems. LNCS, Vol.2154, 2001, pp. 137-151.

[9] K. Onogi, and T.Ushio. Scheduling of Periodic Tasks on a Dynamically Reconfigurable Device Using Timed Discrete Event Systems. IEICE Transactions E89-A(11), 2006, pp. 3227-3234.

[10] E. Mikk, Y. Lakhnech, and M. Siegel. Hierarchical Automata as Model for Statecharts. LNCS, Vol. 1345, 1997, pp. 181-196.

[11] T. A. Henzinger. The Theory of Hybrid Automata. Proc. IEEE Symposium on Logic in Computer Science, 1996, pp. 278-292.

[12] D. Harel. Statecharts: A Visual Formulation for Complex Systems. Sci. Comput. Program. 8(3), 1987, pp. 231-274.

[13] T.A.Henzinger, P.Ho, and H.Wong-Toi A Model Checker for Hybrid Systems. STTT, Vol. 1(1-2), 1997, pp. 110-122.

[14] 南，他．動的再構成可能プロセッサのモデル化，仕様記述とモデル検査．ソフトウェア科学会に投稿中 (論文番号：718)